Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 – Lenguajes de Programación I
Abril-Julio 2006

Carnet: _____

Nombre: _____

Examen II

(40 puntos)

Antes de empezar, revise bien el examen, el cual consta de 4 (CUATRO) preguntas.

Pregunta 0	Pregunta 1	Pregunta 2	Pregunta 3	Total
10 puntos	10 puntos	10 puntos	10 puntos	40 puntos

Pregunta 0 — 10 puntos

Considere la típica construcción de tipos registro (`record`) y tipos arreglo (`array`) utilizada en los lenguajes de la familia de "los Algol" (Pascal, Modula, Ada, etcétera).

Definamos un nuevo tipo `R` como se indica a continuación:

```
type R = record
             c: char;
             b: boolean;
             n: int;
             d: char;
             m: int;
             x: boolean
         end
```

Suponga que el lenguaje está siendo implementado sobre una arquitectura de 32 *bits*, por lo cual cada palabra de memoria, de 4 *bytes*, se encuentra en una dirección múltiplo de 4. Suponga además que la cantidad de espacio que se requiere para objetos de los tipos básicos `boolean`, `char` e `int` es 1 *byte*, 2 *bytes* y 4 *bytes*, respectivamente. Se decide además, por razones de eficiencia de tiempo de ejecución, imponer la restricción de que todo entero (`int`) sea almacenado en 4 *bytes* que constituyan una palabra, esto es, que sea almacenado en una dirección que sea múltiplo de 4. La restricción no aplica para booleanos (`boolean`) ni para caracteres (`char`).

Responda ahora las siguientes preguntas:

(a) Indique cuál es la mínima cantidad de *bytes* de memoria que ocuparía una variable de tipo `R`, suponiendo: (i) que el compilador no se permite reorganizar el orden de los campos de los registros, y (ii) que el compilador sí se permite realizar tal tipo de reorganización (pero, por supuesto, aún respetando la restricción de almacenamiento de enteros arriba señalada).

Explique en ambos casos qué organización interna de memoria (*memory layout*) utiliza para los campos de sus registros tipo `R`.

(b) Indique cuál es la mínima cantidad de *bytes* de memoria que ocuparía una variable de tipo `array [50..100] of R`, para cada uno de los dos casos considerados en **(a)**. Explique sus resultados.

Nota adicional: Para responder a esta pregunta suponga además que el lenguaje permite asignaciones/copias de registros, y que se decidió que éstas deben ser implementadas eficientemente mediante copia de bloques continuos de memoria.

(c) Considere una variable `a` de tipo `array [50..100] of R` y una variable entera `i`, almacenadas respectivamente en las direcciones α y β. Dé una fórmula de cálculo para la dirección de `a[i].x` para cada uno de los dos casos considerados en **(a)**.

Suponga que `a` e `i` fueron almacenadas bajo modelo de valor en las direcciones señaladas. Note además que α y β denotan direcciones de palabras de memoria; esto es, puede asumir que ambas son múltiplos de 4. Utilice $*Z$ para referirse al entero (de 4 *bytes*) contenido en una dirección de palabra Z.

Pregunta 1 — 10 puntos

Considere los dos mecanismos de detección de referencias "colgadas" (*dangling references*) presentados en el texto de M.L. Scott: el uso de tumbas o lápidas (*tombstones*), y el uso de claves de acceso (*locks and keys*).

Para cada uno de estos dos mecanismos, señale qué acciones de bajo nivel deben ser generadas durante la compilación para cada una de las siguientes instrucciones:

- `new(p)`, siendo ésta la instrucción de creación dinámica de objetos y siendo `p` una variable de tipo apuntador a objetos de algún tipo `T`;
- `free(p)`, siendo ésta la instrucción de eliminación dinámica de objetos y siendo `p` como antes;
- `p := nil`, siendo `p` como antes y siendo `nil` la constante que denota el apuntador nulo;
- `p := q`, siendo `p` y `q`, ambas, variables de tipo apuntador a objetos de algún tipo `T`;
- `p^ := x`, siendo `p` como antes, siendo "`^`" el operador de indirección de apuntadores y siendo `x` una variable de tipo `T` (suponga que el lenguaje permite la asignación, mediante copia de valores, de objetos de cualquier tipo `T`).

Note que debe analizar las cinco instrucciones independientemente, sin asumir que éstas ocurren secuencialmente en ningún orden particular. Note también que el término "objeto" está siendo utilizado simplemente como sinónimo de "cosa", y no en el sentido de "programación orientada por objetos" ("*object-oriented programming*").

Para su respuesta, suponga que las variables `p`, `q` y `x` están almacenadas en las direcciones α, β y γ, respectivamente. Note que α, β y γ denotan direcciones, y utilice $*Z$ para referirse al contenido de cualquier dirección Z. Puede suponer además que tanto un apuntador como un entero ocupan 4 *bytes* y que se está trabajando sobre una arquitectura de 32 *bits*.

La descripción de las acciones de bajo nivel generadas puede ser dada en lenguaje natural, utilizando apropiadamente la notación $*Z$ propuesta. Cuando se deba generar un error dinámico, indique claramente si éste corresponde a una referencia "colgada" o simplemente corresponde a intentar indireccionar la referencia nula, o a cualquier otra condición indeseada.

Pregunta 2 — 10 puntos

Considere el siguiente programa escrito en lenguaje C:

```
void foo ()
{
    int i;
    printf ("%d ", i++);
}

main ()
{
    int j;
    for (j = 1; j <= 10; j++)
        foo ();
}
```

Note que el programador olvidó inicializar la variable local `i` en la subrutina `foo`.

Para algunas implementaciones del lenguaje, el programa siempre generará determinísticamente la salida `0 1 2 3 4 5 6 7 8 9`. Esto es, toda ejecución del programa se comportará de esta manera. Sugiera una explicación para esto.

Dé además otras dos alternativas posibles de comportamiento que este programa podría presentar sobre otras implementaciones del lenguaje, una de las cuales debe ser determinística y la otra no-determinística. Esto es, sobre una de las alternativas de implementación del lenguaje todas las ejecuciones del programa deben generar la misma salida, y sobre la otra alternativa de implementación del lenguaje distintas ejecuciones del programa pueden generar distintas salidas. Señale a qué se debe el comportamiento observado en ambos casos.

Nota: Si le conviene, ignore el requerimiento usual de intentar maximizar la eficiencia de tiempo de ejecución que se asocia a la "filosofía" de lenguaje C.

Pregunta 3 — 10 puntos

Construya un pequeño programa en Java que cuente con un procedimiento (no función), esto es, un método estático sin valor de retorno, que reciba un único parámetro de tipo entero y que se comporte de cinco maneras diferentes (esto es, que dé cinco salidas de escritura diferentes) dependiendo del mecanismo de pasaje de parámetros utilizado: valor, resultado, valor/resultado, referencia y nombre.

Para el caso de pasaje por resultado, suponga que el parámetro formal es inicializado en cero.

Puede utilizar lenguaje C o cualquier otro lenguaje imperativo de su escogencia, en lugar de Java, si así Ud. lo prefiere.